LAMBDA

Getting Started with IVI-COM and Python for the Lambda Genesys Power Supply

1. Introduction

This is a guide to writing programs using the Python language with the Lambda IVI-COM drivers. Python is praised for being simple but powerful. It is open-source and may be installed at no cost. Many standard libraries and 3d party extensions are available.

IVI (Interchangeable Virtual Instrument) is a design by the global IVI Foundation to simplify the way equipment is used with computer automation programs. This guide will allow you to remotely control the Genesys power supply using the RS-232/485, IEEE-488 or LAN interfaces.

2. Table of Contents

Table of Figures

3. Requirements

A. Windows Operating System: This tutorial is written using Windows XP SP2. Because of the requirement for VISA hardware drivers and COM objects, this tutorial cannot say that other operating systems are supported.

B. Python Software: This tutorial requires the following downloads:

1. Python version 2.5.2.

Earlier versions have not been tested. The "comtypes" package may not be compatible with earlier versions of Python.

http://www.python.org/download/

Used here: python-2.5.2.msi

2. PythonWin build 211

Python extensions for Windows. This is a Windows graphical development environment for compiling Python projects.

http://sourceforge.net/project/platformdownload.php?group_id=78018

Used here: pywin32-211.win32-py2.5.exe

3. comtypes version 0.4.2

This is a 3d party Python extension to allow early binding of COM objects with unknown interfaces.

http://pypi.python.org/pypi/comtypes/0.4.2

Used here: comtypes-0.4.2.win32.exe

C. IVI Drivers: The Lambda Genesys IVI driver package must be installed. It is available as a download from:

http://www.lambda-hp.com/product_html/genesys1u.htm

Click on the "Documentation → Download Drivers" link to register.

Install the file "LambdaGenPS.2.x.x.x.msi" on your computer

D. VISA Drivers: The IVI drivers are built upon a layer of hardware drivers called VISA (for "Virtual Instrument Software Architecture"). These drivers may be obtained from a variety of companies who support test and measurement.

The runtime versions of the VISA drivers are acceptable, but you will not have the communications utilities or the IVI configuration manager to create logical device names.

E. Electrical Interface: almost all personal computers have an RS-232 port or a local area network (LAN) port. If your Genesys power supply uses the RS-485 or IEEE-488 (IEMD option) ports, you will need to install these interface cards into your computer or laptop.

F. Prior Experience with Python: This tutorial assumes very little prior experience with Python programming. It will do nothing more than use the PythonWin Interactive screen to create the Genesys supply IVI-COM object and send a few commands.

4. Installation Instructions

Download the files listed in the "Requirements" section above. Then:

A. Install the Python Program

You will create a folder such as:

C:\Python25\…

B. Install the PythonWin Program

You will create a folder and program such as:

C:\Python25\Lib\site-packages\pythonwin\ Pythonwin.exe

C. Install the Python "comtypes" Extension

You will create a folder such as:

C:\Python25\Lib\site-packages\comtypes\…

D. Install the Genesys IVI Drivers

You will create a folder and library such as:

C:\Program Files\IVI\Bin\ LaGen.dll

E. Install the VISA Drivers

You will create a folder such as:

C:\VXIPNP\… or C:\Program Files\VISA\… or other

F. Additional Help

A discussion of the Python interface for COM objects may be found at:

http://starship.python.net/crew/theller/comtypes/

Information on the IVI-COM drivers, including the commands and settings, is installed by the Lambda Genesys IVI driver package. A "Getting Started" and language reference Help is included. This help file may be opened through the Start menu as shown below:

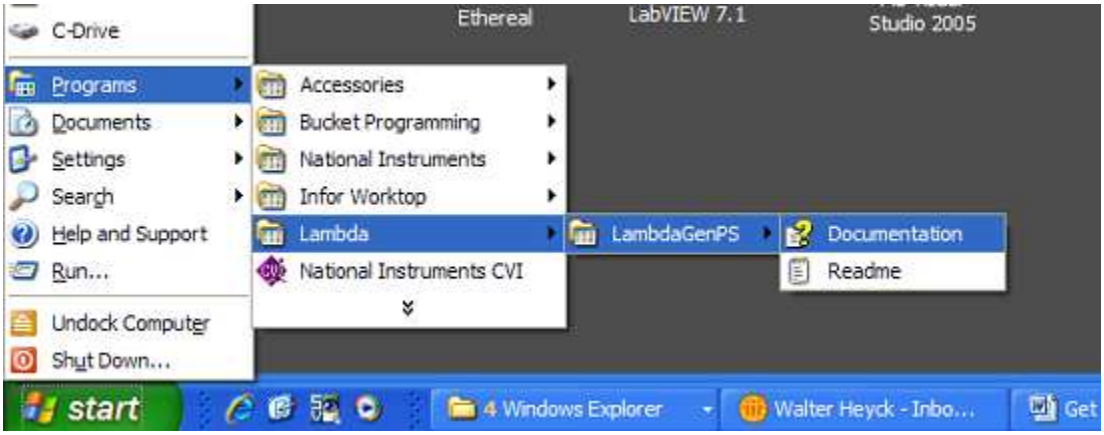

Figure 1. Launching IVI "Help"

Below is a sample of the Help file's contents:

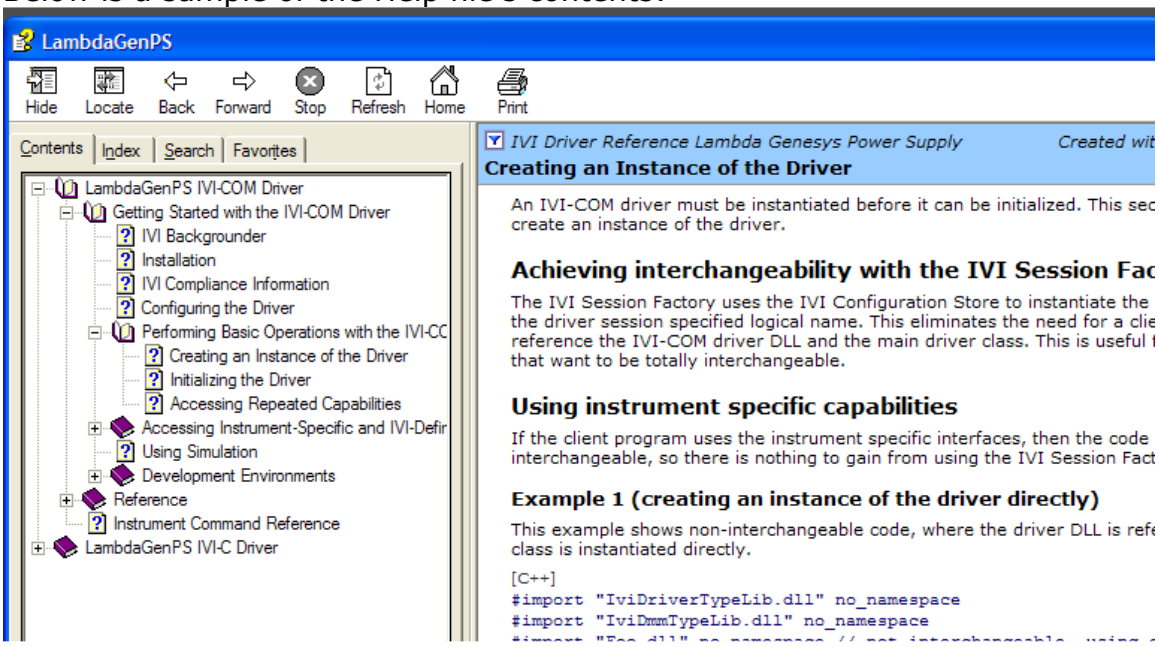

Figure 2. Sample of IVI "Help"

5. Your First Program Using Python

A. Launch PythonWin.

On the Windows Start menu, launch PythonWin as shown below.

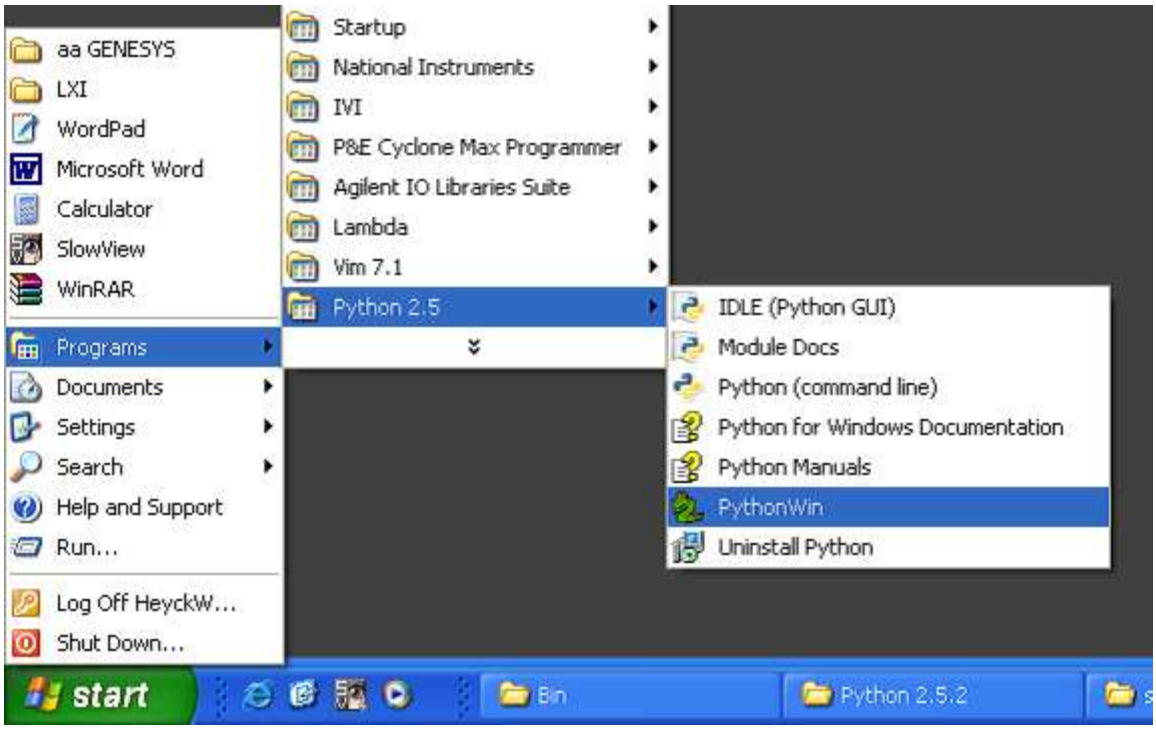

Figure 3. Launching PythonWin

B. *Create the Power Supply Object*

At the Python ">>>" prompt, type the lines:

```
>>> from comtypes.client import CreateObject

>>> PS1 = CreateObject( "LambdaGenPS.LambdaGenPS")
```

"PS1" is an arbitrary name created to identify the reference to the power supply object. You may choose any name.

The first time you run the "CreateObject" command, a list of "Generating comtypes…." notices are displayed.

After this, when you begin to type commands to the "PS1" object, you will see the Intellisense list boxes appear as shown below:

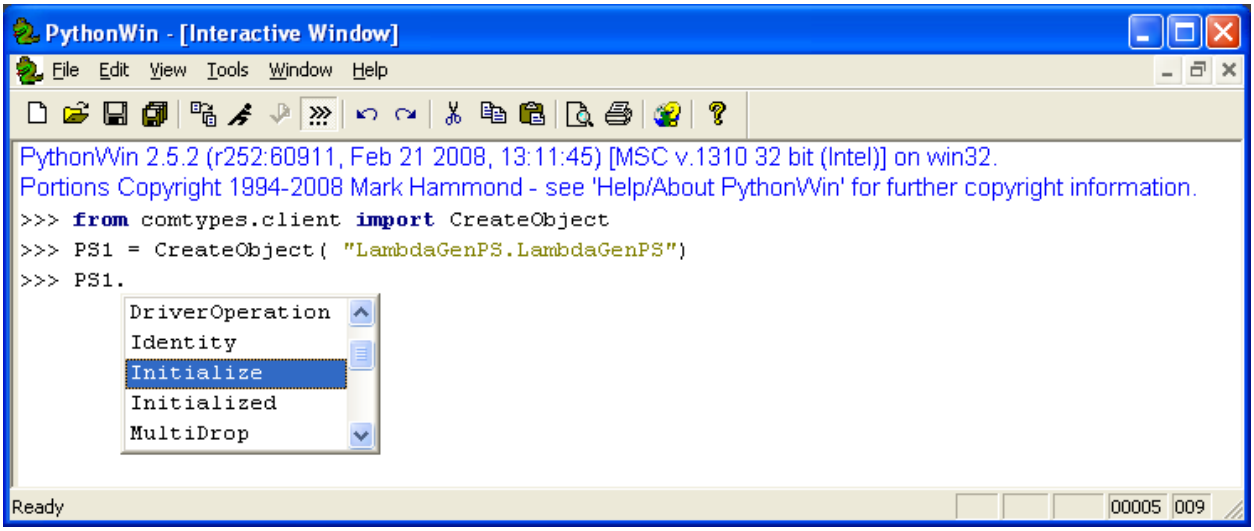

Figure 4. Intellisense

C. *Initialize the Genesys Power Supply*

Once the COM object is created, the first step is to initialize it. This step is used to select the instrument (including what bus it is using and it's address) and do other things. In the Figure 5 example below, the step:

```
PS1.Initialize( "GPIB::6::INSTR", False, False, "")
```

will assign a name "PS1" and open communication with a Genesys power supply at IEEE address 6. This step will also verify that communication can be established with the power supply.

See section 6 for much more information on the Initialize step.

D. Communicate with the Genesys Power Supply

Once the Genesys IVI-COM object is initialized (named "PS2" in the figure below), you can start communicating with it.

The Intellisense (see Figure 4) guides you through the choices of properties and methods for the "PS2" object. More specific details about them may be found in the IVI online "Help" as shown in Figure 2.

```
PythonWin 2.5.2 (r252:60911, Feb 21 2008, 13:11:45) [MSC v.1310 32 bit (Intel)] on win32.
Portions Copyright 1994-2008 Mark Hammond - see 'Help/About PythonWin' for further copyright information.
>>> from comtypes.client import CreateObject
>>> PS2 = CreateObject( "LambdaGenPS.LambdaGenPS")
>>> PS2.Initialize( "GPIB::6::INSTR", True, True, "")
0
>>> PS2.Output.VoltageLimit = 5
>>> PS2.Output.CurrentLimit = 1
>>> PS2.Output.Enabled = True
>>> PS2.Output.MeasureVoltage( )
5.0019999999999998
>>> PS2.Identity.InstrumentModel
u'GEN30-25-IEMD'
>>> PS2.Utility.Reset( )
0
>>> PS2.Close( )
0
>>> PS2.Release( )
0L
>>>
```

Figure 5. Example Python Session

6. The "Initialize" Step

Once the COM object is created, the first step is to initialize it. An example is shown in the third command in Figure 5:

```
PS1.Initialize( "GPIB::6::INSTR", False, False, "")
```

This important step will:

1. Use a resource descriptor to:
   a. Identify the port type (IEEE, RS-232/485 or LAN).
   b. Describe the instrument's address on the bus.
2. Optionally, verify the supply's identity.
3. Optionally, reset the supply.
4. Optionally, set IVI options such as simulation and error checking.

The initialize step will also verify that communication can be established with the power supply.

A. Parameter 1: Select the Instrument

This is a string name that specifies the Genesys port type and address. The name may be one of:

1. VISA resource descriptor.

If you only have the VISA runtime engine installed, you must use this form.

Examples of use are:

- **For the IEEE bus:**

  `"GPIB::6::INSTR"` for supply at IEEE address 6.

- **For the RS-232/485 bus:**

  `"ASRL1::INSTR"` for supply connected to the computer's serial port 1. The address and Baud rate are given in the option string, which is the fourth parameter.

  `"COM1"` is an alias which can usually be used for the computer's serial port 1. The address and Baud rate are given in the option string, which is the fourth parameter.

- **For the LAN bus:**

  `"TCPIP::192.168.47.92::INSTR"` for a supply connected to a local network. The supply's IP address, as read on it's front panel, is 192.168.47.92.

  `"TCPIP::GENH30-25-827::INSTR"` for a supply connected to a local network. The supply's hostname, as read on it's web LAN settings page, is GENH30-25-827.

2. **IVI Logical Name.**

   If you have a utility that can manage the IVI configuration, you can use Python to initialize an IVI logical device. This allows the use of the IVI interchangeability feature which means you can swap one manufacturer's power supply for another's without changing the Python program.

   IVI configuration utilities include the National Instruments Measurement and Automation Explorer (NI-MAX), the Agilent VEE Pro Instrument Manager and others.

B. Parameter 2: Identity Query

If this parameter is set to "`True`", then the Initialize method will read the power supply's identity string (" `*IDN?`") and verify that a Genesys supply is being initialized.

C. Parameter 3: Reset

If this parameter is set to "`True`", then the Initialize method will reset the power supply. This includes setting the voltage and current to zero and the output off. Other settings are set to the default; see the User Manual.

D. Parameter 4: Option String

The fourth parameter may be empty, empty double quotes (a NULL string), or one of the following. Other parameter settings are available, only the most popular are shown below. See IVI Help, shown in Figure 1, for more options.

Option String **= "Simulate=True"**

Enabling the Simulate mode allows you to run the Python program even if no power supply is actually connected to the system. Commands are simulated and responses are fixed default values.

```
>>> from comtypes.client import CreateObject
>>> GenPS = CreateObject( "LambdaGenPS.LambdaGenPS")
>>> GenPS.Initialize( "TCPIP::192.168.14.57::INSTR",False,False,"Simulate=True")
0
>>> GenPS.Output.Enabled = True
>>> GenPS.Output.Enabled
True
>>> GenPS.Identity.InstrumentModel
u'GENvvv-aaa-LAN'
>>> GenPS.Output.MeasureVoltage( )
1.5
>>>
```

Figure 6. Simulating an Instrument

Option String **= "QueryInstrStatus=True"**

This will cause an " *ESR?" query to be automatically sent after every command. If a command error, or other error, is returned, a COM exception will be thrown. See section 7 below for an example.

```
>>> PS2.Initialize( "GPIB::6::INSTR", True, True, "QueryInstrStatus=True")
0
```

Figure 7. Enabling Automatic Error Query

Multiple Option Strings

Multiple options are set by using a comma separator between the option parameter strings. Below is an example of simulating an RS-232 power supply:

```
>>> from comtypes.client import CreateObject
>>> PS1 = CreateObject( "LambdaGenPS.LambdaGenPS")
>>> PS1.Initialize(
"COM1",False,False,"Simulate=True,DriverSetup=SerialAddress=6,BaudRate=9600")
0
>>> PS1.Identity.InstrumentFirmwareRevision
u'Sim2.0.3.0'
>>> PS1.Output.VoltageLimit = 100
>>> PS1.Output.MeasureVoltage( )
1.5
>>>
```

Figure 8. Enabling a Simulated RS-232 Instrument

7. Catching Errors with Exceptions

When an IVI-COM method is called, it will return with a non-zero HRESULT value that describes a type of COM error. It will also throw a COMexception event. By default, COMexceptions will only report errors detectable by the computer, including unopened ports, cable disconnections or no responses from the power supply.

By modifying the "Initialize" step (see section 6), the COMexception can be thrown for instrument errors such as an illegal command or a setting out of range. To do this, change the fourth parameter (Option String) in the "Initialize" step to:

"QueryInstrStatus=True"

This will cause the IVI driver check after every command to make sure the command was accepted by the power supply. If a command error occurred then a COMexception will be thrown.

This method is shown in the Python session below. Notice the traceback after trying to set the supply above its rating, to one thousand volts:

```
PythonWin 2.5.2 (r252:60911, Feb 21 2008, 13:11:45) [MSC v.1310 32 bit (Intel)] on win32.
Portions Copyright 1994-2008 Mark Hammond - see 'Help/About PythonWin' for further copyright information.
>>> from comtypes.client import CreateObject
>>> PS1 = CreateObject( "LambdaGenPS.LambdaGenPS")
>>> # Notice, in line below, no spaces after commas
>>> PS1.Initialize( "GPIB0::6::INSTR",False,False,"QueryInstrStatus=True")
0
>>> PS1.Output.Enabled = True
>>> PS1.Output.VoltageLimit = 1000
Traceback (most recent call last):
  File "<interactive input>", line 1, in <module>
  File "C:\Python25\Lib\site-packages\comtypes\__init__.py", line 231, in
__setattr__
    value)
COMError: (-2147192831, None, (u'LambdaGenPS: Instrument error detected.  Use
ErrorQuery() to determine the error(s).', u'LambdaGenPS.LambdaGenPS.1', None, 0,
None))
>>>
```

Figure 9. Using COMexception to Catch Errors

Although this COMexception method will display all types of errors at any time, this example does not show how to programmatically handle the exception. This example also does not show how to report the instrument error code or description.

8. Catching Errors with SYST:ERR?

For a program to be reliable, it is necessary to check for instrument errors. The COMexception example in the above section will catch an error event, but it does not report the instrument error code or description and it does not give any way to handle an error.

A solution to handle instrument errors is shown below. The power supply has a system error queue that reports error codes and descriptions in a human readable string. The IVI-COM DirectIO method is used to send and read SCPI commands directly to the power supply.

```
PythonWin 2.5.2 (r252:60911, Feb 21 2008, 13:11:45) [MSC v.1310 32 bit (Intel)] on win32.
Portions Copyright 1994-2008 Mark Hammond - see 'Help/About PythonWin' for further copyright information.
>>> from comtypes.client import CreateObject
>>> PS1 = CreateObject( "LambdaGenPS.LambdaGenPS")
>>> # In next step, do not enable driver to do QueryInstrStatus
>>> PS1.Initialize( "GPIB0::6::INSTR",False,False, )
0
>>> # Enable the error queue and clear it
>>> PS1.System.DirectIO.WriteString( "SYST:ERR:ENAB")
0
>>> PS1.System.DirectIO.WriteString( "SYST:ERR?")
0
>>> PS1.System.DirectIO.ReadString( )
u'0,"No error"'
>>> # Now, send an illegal setting
>>> PS1.Output.CurrentLimit = 1000
>>> PS1.System.DirectIO.WriteString( "SYST:ERR?")
0
>>> PS1.System.DirectIO.ReadString( )
u'-222,"Data out of range;address 06"'
>>>
```

Figure 10. Using SYST:ERR to Catch Errors

Note that setting "CurrentLimit = 1000" is not an error for the IVI driver so no COMexception was thrown. It is, however, an error for the power supply so the system error queue reported it. The response from the ReadString may be managed using the normal Python string functions.

9. Programming with the RS-232/485 Port

Writing IVI-COM programs for the RS-232/485 ports is the same as the example above for the IEEE interface. The only change is in the initialize option string

A. RS-232 Initialize

For the IEEE Initialize, the entire connection is defined in the VISA resource name string. For the RS-232/485 port, The VISA name only selects the computer's COM port (COM1, COM2 etc.). The Genesys address and the Baud rate are specified in the fourth parameter of the call to Initialize. This fourth parameter is an option string.

In the example below, the following settings are shown:

- "`ASRL1`" = the computer's COM1 port is selected
- "`DriverSetup=`" is the required start of the RS-232 string.
- "`SerialAddress=6`" is the Genesys front panel address setting.
  It may be a number from 0 to 30.
- "`BaudRate=9600`" is the Genesys front panel serial bits-per-second.
  It may be 1200, 2400, 4800, 9600 or 19200

```
PythonWin 2.5.2 (r252:60911, Feb 21 2008, 13:11:45) [MSC v.1310 32 bit (Intel)] on win32.
Portions Copyright 1994-2008 Mark Hammond - see 'Help/About PythonWin' for further copyright information.
>>> from comtypes.client import CreateObject
>>> PS1 = CreateObject( "LambdaGenPS.LambdaGenPS")
>>> # In next step, "ASRL1::INSTR" is the generic name for "COM1"
>>> PS1.Initialize( "ASRL1::INSTR",True,True,"DriverSetup=SerialAddress=6,BaudRate=9600")
0
>>> PS1.Identity.InstrumentManufacturer
u'LAMBDA'
>>> PS1.Utility.Reset( )
0
>>> PS1.Output.CurrentLimit = 5
>>> PS1.value.Output.CurrentLimit
5.0
>>>
```

Figure 11. Initializing the RS-232 Port

B. Catching Errors

For the serial RS-232 port, any setting out of range or other illegal command will generate a COMexception. There is no need to Initialize the driver with `"QueryInstrStatus=True"`.

10. Programming with the LAN Port

Writing IVI-COM programs for the LAN port is the same as the example above for the IEEE interface. The only change is in the initialize option string.

A. LAN Initialize with IP Address

Like the IEEE Initialize, the entire LAN connection is defined in the VISA resource name string. Below is an example if the power supply auto-IP address is "169.254.7.58".

```
PythonWin 2.5.2 (r252:60911, Feb 21 2008, 13:11:45) [MSC v.1310 32 bit (Intel)] on win32.
Portions Copyright 1994-2008 Mark Hammond - see 'Help/About PythonWin' for further copyright information.
>>> from comtypes.client import CreateObject
>>> PS1 = CreateObject( "LambdaGenPS.LambdaGenPS")
>>> PS1.Initialize( "TCPIP::169.254.7.58::INSTR",False,False, )
0
>>> # Read the network hostname
>>> PS1.System.DirectIO.WriteString( "SYST:COMM:LAN:HOST?")
0
>>> PS1.System.DirectIO.ReadString( )
u'GEN60-85-181'
>>>
```

B. LAN Initialize with Hostname

If the LAN power supply is connected through a network server, it's hostname may be registered by the NBNS or DNS services. In this case, the hostname may be used in the VISA resource name string instead of the IP address. Below is an example if the power supply hostname is "GENH30-25-827".

```
PythonWin 2.5.2 (r252:60911, Feb 21 2008, 13:11:45) [MSC v.1310 32 bit (Intel)] on win32.
Portions Copyright 1994-2008 Mark Hammond - see 'Help/About PythonWin' for further copyright information.
>>> from comtypes.client import CreateObject
>>> PS2 = CreateObject( "LambdaGenPS.LambdaGenPS")
>>> PS2.Initialize( "TCPIP::GENH30-25-827::INSTR",True,True, )
0
>>> PS2.Identity.InstrumentModel
u'GENH30-25-LAN'
>>>
```